DSD HW3 Report

B11901132 林軒霆

1. Cycle time & memory devices

   Since this homework does not require optimizing design with respect to any metrics, I used the original 10 ns as my cycle time.

   The inferred memory devices during synthesis for both designs are shown below:

```
Inferred memory devices in process
        in routine cache line 210 in file
                '/home/raid7_2/userb11/b11132/DSD/DSD_HW3/cache_dm.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|      state_reg      | Flip-flop |   3   |  Y  | N  | N  | N  | N  | N  | N  |
|      dirty_reg      | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|       tag_reg       | Flip-flop |  200  |  Y  | N  | N  | N  | N  | N  | N  |
|      valid_reg      | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|     blocks_reg      | Flip-flop | 1024  |  Y  | N  | N  | N  | N  | N  | N  |
===============================================================================
```

```
Inferred memory devices in process
        in routine cache line 219 in file
                '/home/raid7_2/userb11/b11132/DSD/DSD_HW3/cache_2way.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|      state_reg      | Flip-flop |   3   |  Y  | N  | N  | N  | N  | N  | N  |
|       tag_reg       | Flip-flop |  200  |  Y  | N  | N  | N  | N  | N  | N  |
|      dirty_reg      | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|       lru_reg       | Flip-flop |   4   |  Y  | N  | N  | N  | N  | N  | N  |
|      valid_reg      | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|     blocks_reg      | Flip-flop | 1024  |  Y  | N  | N  | N  | N  | N  | N  |
===============================================================================
```

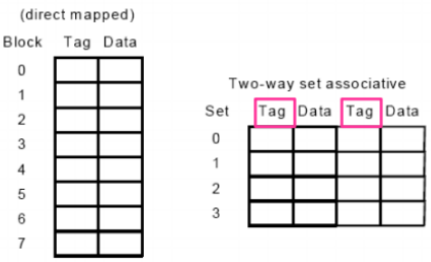
2. General specification

   (a.) Direct-mapped cache

   According to the HW specification, I used 8 blocks with 4 words each. I used the write-back policy to update memory with dirty blocks.

   (b.) Two-way associative cache

   According to the HW specification, I used 4 sets, each with 2 blocks. The cache uses a write-back policy and employs the Least Recently Used (LRU) replacement strategy.



3. Finite state machine

In both of my designs, I used the following four states for my FSM:

(a.) S_IDLE

- Default state for my cache
- Handles cache hits directly
- For misses, check if target block is dirty and needs writing back

(b.) S_WRITE_BACK

- Activates when a miss occurs on a valid, dirty block
- Writes cache data back to memory
- Transitions to S_READ_MEM once write-back is complete

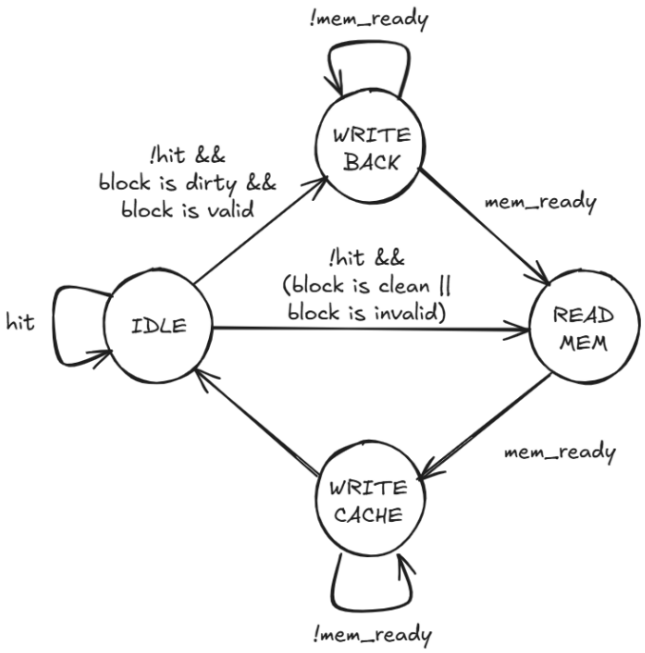
(c.) S_READ_MEM

- Reads the required block from memory
- Activates after S_WRITE_BACK or directly from S_IDLE for clean/invalid blocks
- Transitions to S_WRITE_CACHE once the memory read is complete

(d.) S_WRITE_CACHE

- Updates the cache with the new data from memory
- For write operations, also updates with the processor's write data
- For read operations, prepares the requested data for the processor
- Always transitions back to S_IDLE

A diagram of the FSM with the control signals that determine state transition:



4. Performance evaluation

I modified the testbench file to get statistics of read/write accesses/misses etc., the following table summarizes the performance of the two cache designs.

	Direct-mapped	Two-way associative
Total cycles	13572	8964
Stall cycles	10496	5888
Read accesses	2048	2048
Read misses	1280	512
Read miss rate	62.50 %	25.00 %
Write accesses	1024	1024
Write misses	256	256
Write miss rate	25.00 %	25.00 %
Overall miss rate	50.00 %	25.00 %

5. Comparison of two architectures

There are several aspects we can compare between the two architectures

(a.) Overall performance

The two-way associative cache completes the test in significantly fewer cycles, resulting in a reduction of execution time.

(b.) Stall

The two-way cache reduces both the absolute number of stalls and the stall rate.

(c.) Read performance

The two-way cache dramatically reduces read misses compared to direct-mapped cache. This shows how the additional flexibility of having two possible locations for each cache block helps reduce conflict misses.

(d.) Write performance

Both designs have the same write miss rate, and could be explained by the specific access patterns in the testbench. The write phase of the test likely doesn't trigger many conflict misses that would benefit from the two-way associative design.

6. Bonus

   I used the LRU policy instead of randomly choosing blocks during write back to memory, which was implemented using a register to keep hold of the latest written block for each set.

7. Discussion & experiences

   HW3 and together with the Exercise finished not long ago showcased the power of FSM to model complex systems, which I did not understand just a few weeks ago before taking these assignments. I found that adopting a clean coding style and designing a well-structured FSM greatly enhanced code readability, making the development process both efficient and satisfying.